

7.

Pascalmaskinen

Indhold:

- 7.1 Grundkonstruktioner
 - a) Note: Implementering af konstanter og variable datafelter
 - b) Note: Implementering af selektion og iteration
 - c) Note: Implementering af tabeller med index-adressering
 - d) Note: Implementering af sekventiel tabelgennemløb
- 7.2 Procedurer og funktioner med parametre
 - a) Note: Implementering af procedure og funktioner med brug af stak til parametre.
- 7.3 Systemkald direkte fra pascal
 - a) Note (ikke medtaget endnu)
- 7.4 Assembler direkte fra pascal
 - a) Note (ikke medtaget endnu)
- 7.5 Komplet programeksempel - fra pascal til assembler
 - a) Programeksempel i pascal
 - b) Programeksempel i pascal med inline assembler
 - c) Programeksempel i assembler

7.1.a

Implementering af konstanter og variable datafelter

Bjørk Busch

Initiering på oversættelsestidspunktet:

PASCAL

CONST

```
KONSTANT = 15;
STARTTAL: INTEGER = 10;
```

ASSEMBLER

```
KONSTANT EQU 15
STARTTAL DW 10
```

Værditilskrivning på kørselstidspunktet:

PASCAL

VAR {main}

```
ITAL : INTEGER;
ITAL2 : INTEGER;
BTAL : BYTE;
BTAL2 : BYTE;
TEGN : CHAR;
TEGN2 : CHAR;
TEKST : STRING[5];
ITABEL : ARRAY[1..5]
OF INTEGER;
```

ASSEMBLER

{datasegment}

ITAL	DW	?
ITAL2	DW	?
BTAL	DB	?
BTAL2	DB	?
TEGN	DB	?
TEGN2	DB	?
TEKST	DB	6 DUP (?)
ITABEL	DW	5 DUP (?)

Tilskrivning med konstant:

ITAL := 5;

MOV ITAL, 5

TEKST := 'ABC';

MOV TEKST+0, 3 ; længde
 MOV TEKST+1, 'A'
 MOV TEKST+2, 'B'
 MOV TEKST+3, 'C'

```
ITABEL[1] := 10;
ITABEL[2] := 20;
ITABEL[3] := 30;
ITABEL[4] := 40;
ITABEL[5] := 50;
```

MOV ITABEL+0, 10
 MOV ITABEL+2, 20
 MOV ITABEL+4, 30
 MOV ITABEL+6, 40
 MOV ITABEL+8, 50

Tilskrivning med variabel:

ITAL := ITAL2 ;	MOV AX, ITAL2 MOV ITAL, AX
BTAL := BTAL2 ;	MOV A1, BTAL2 MOV BTAL, A1
TEGN := TEGN2 ;	MOV A1, TEGN2 MOV TEGN, A1
ITABEL[3] := ITAL ;	MOV AX, ITAL MOV ITABEL+6, AX

Tilskrivning med simpel typekonvertering:

TALB := ORD(TEGN) ;	MOV A1, TEGN MOV TALB, A1
TEGN := CHR(BTAL) ;	MOV A1, BTAL MOV TEGN, A1
ITAL := BTAL ;	MOV AL, BTAL CBW MOV ITAL, AX
BTAL := ITAL ;	MOV AX, ITAL MOV BTAL, AL

; AX:=AL
; -kontrol

Tilskrivning med simpel 16-bits beregning:

ITAL := ITAL + ITAL2 ;	MOV AX, ITAL ADD AX, ITAL2 MOV ITAL, AX
ITAL := ITAL - ITAL2 ;	MOV AX, ITAL SUB AX, ITAL2 MOV ITAL, AX
ITAL := ITAL * ITAL2 ;	MOV AX, ITAL MOV BX, ITAL2 IMUL BX MOV ITAL, AX
ITAL := ITAL DIV ITAL2 ;	MOV AX, ITAL CWD MOV BX, ITAL2 IDIV BX MOV ITAL, AX
ITAL := ITAL MOD ITAL2 ;	MOV AX, ITAL CWD MOV BX, ITAL2 IDIV BX MOV ITAL, DX

7.1.b

Implementering af selektion og iteration

Bjørk Busch

Simpel IF sætning:

PASCAL.

A og B er 2 integer variable:

```
IF (A < B) THEN BEGIN
  .....
END
ELSE BEGIN
  ...
END;
```

Omskrevet til assembler:

IIFTST:	MOV	AX, A			
	CMP	AX, B			
	JNL	ELSEBEG	eller	JL	IFBEG
			og	JMP	ELSEBEG
IFBEG:				
				
	JMP	IFEND			
ELSEBEG:				
				
IFEND:					

PASCAL.

A, B og C er 3 integer variable:

```
IF ((A + B) = C) THEN BEGIN
  .....
END
ELSE BEGIN
  ...
END;
```

Omskrevet til assembler:

IFCALC:	MOV	AX, A			
	ADD	AX, B			
IIFTST:	CMP	AX, C			
	JNE	ELSEBEG	eller	JE	IFBEG
			og	JMP	ELSEBEG
IFBEG:				
				
	JMP	IFEND			
ELSEBEG:				
				
IFEND:					

Sammensat IF sætning med AND:**PASCAL.**

A,B og C er 3 integer variable:

```
IF (A < B) AND (B > C) THEN BEGIN
  .....
END
ELSE BEGIN
  ...
END;
```

Omskrevet til assembler:

IIFTST:	MOV	AX,A			
	CMP	AX,B			
	JNL	ELSEBEG	eller	JL	TEST2
			og	JMP	ELSEBEG
IIFTST2:	MOV	AX,B			
	CMP	AX,C			
	JNG	ELSEBEG	eller	JG	IFBEG
			og	JMP	ELSEBEG
IFBEG:				
				
	JMP	IFEND			
ELSEBEG:				
				
IFEND:					

Sammensat IF sætning med OR:**PASCAL.**

A,B og C er 3 integer variable:

```
IF (A < B) OR (B > C) THEN BEGIN
  .....
END
ELSE BEGIN
  ...
END;
```

Omskrevet til assembler:

IIFTST:	MOV	AX,A			
	CMP	AX,B			
	JL	IFBEG			
	MOV	AX,B			
	CMP	AX,C			
	JNG	ELSEBEG	eller	JG	IFBEG
			og	JMP	ELSEBEG
IFBEG:				
				
	JMP	IFEND			
ELSEBEG:				
				
IFEND:					

Simpel WHILE sætning:**PASCAL.**

A og B er 2 integer variable:

```
WHILE (A < B) DO BEGIN
  .....
END;
```

Omskrevet til assembler:

```
WHILE:    MOV      AX,A
          CMP      AX,B
          JNL      ENDWHILE
WHILEBEG: .....
          .....
          JMP      WHILE
ENDWHILE:
```

Simpel REPEAT sætning:**PASCAL.**

A og B er 2 integer variable:

```
REPEAT
  .....
UNTIL (A < B);
```

Omskrevet til assembler:

```
REPEAT:   .....
          .....
UNTIL:   MOV      AX,A
          CMP      AX,B
          JNL      REPEAT
```

Simpel FOR sætning hvor tæller ikke anvendes:**PASCAL.**

```
FOR I := 1 TO 10 DO BEGIN
  .....
END;
```

Omskrevet til assembler:

```
FORINIT: MOV      CX,10
FORBEG:  .....
          .....
FORNEXT: LOOP    FORBEG    eller
          og           DEC CX + JNZ
                      JNZ    FORBEG
```

Gennel model for FOR-konstruktion der tæller op og hvor tæller kan anvendes:

PASCAL.

I er en integer.

```
FOR I := START TO STOP DO BEGIN
    .....
END;
```

Omskrevet til assembler:

FORINIT:	MOV	AX, START
	MOV	I, AX
FORTEST:	MOV	AX, I
	CMP	AX, STOP
	JG	FOREXIT
FORBEG:	PUSH	I ; SAVE af I - kan undværes
	
	
FOREND:	POP	I ; --- " " ---
	INC	I
	JMP	FORTEST
FOREXIT:		

Gennel model for FOR-konstruktion der tæller ned og hvor tæller kan anvendes:

PASCAL.

I, Start og Stop er integer.

```
FOR I := Start DOWNTO Stop DO BEGIN
    .....
END;
```

Omskrevet til assembler:

FORINIT:	MOV	AX, START
	MOV	I, AX
FORTEST:	MOV	AX, I
	CMP	AX, STOP
	JL	FOREXIT
FORBEG:	PUSH	I ; SAVE af I - kan undværes
	
	
FOREND:	POP	I ; --- " " ---
	DEC	I
	JMP	FORTEST
FOREXIT:		

Sammensat IF sætning med boolske variable og OR:**PASCAL.**

A,B og C er 3 byte's med der kun kan indeholde 0 og 1
(svarende til pascal boolean) :

```
IF (A = B) OR (B > C) THEN BEGIN
  .....
END
ELSE BEGIN
  ...
END;
```

Omskrevet til assembler:**BOOLSK - 2 test løsning uden CMP, SUB, ADD m.m.:**

IIFTST:	MOV	AX,A				
	XOR	AX,B				
	JZ	IFBEG	B C		B > C	
	MOV	AX,C	-----	-----		
	NOT	AX	0 0		0	
	AND	AX,B	0 1		0	
	JZ	ELSEBEG	1 0		1	
IFBEG:		1 1		0	
		-----	-----		
	JMP	IFEND			B AND (NOT C)	
ELSEBEG:					
					
IFEND:						

REN BOOLSK løsning:

IIFTST:	MOV	AX,A				
	XOR	AX,B				
	NOT	AX			; AX := (A = B)	
	MOV	BX,AX			; Gem	
	MOV	AX,C				
	NOT	AX				
	AND	AX,B			; AX := (B > C)	
	OR	AX,BX			; Saml. med OR	
IFBEG:	JZ	ELSEBEG				
					
					
ELSEBEG:	JMP	IFEND				
					
IFEND:					

7.1.c

Implementering af tabeller med index-adressering

Bjørk Busch

På assemblerniveau arbejdes ikke med elementnumre men med celleadresser, og index skal derfor omregnes til celleantal.

1. dimensional tabel

PASCAL

```
VAR {main}
```

```
ITABEL : ARRAY [1..5]
          OF INTEGER;
INDEX    : INTEGER;
ITAL     : INTEGER;
```

```
ITABEL [INDEX] := ITAL;
```

```
ITAL := ITABEL [INDEX];
```

```
ITAL := ITABEL [INDEX+2];
```

ASSEMBLER

```
{databsegment}
```

```
ELMLEN EQU 2
STARTIDX EQU 1
ITABEL DW 5 DUP (?)
```

```
INDEX DW ?
ITAL DW ?
```

```
MOV AX, ITAL
PUSH AX
MOV AX, INDEX
SUB AX, STARTIDX
MOV SI, ELMLEN
MUL SI
MOV SI, AX
POP AX
* MOV ITABEL[SI], AX
```

* kan erstattes af:
`LEA BX, ITABEL
MOV [BX+SI], AX`

```
MOV AX, INDEX
SUB AX, STARTIDX
MOV SI, ELMLEN
MUL SI
MOV SI, AX
* MOV AX, ITABEL[SI]
MOV ITAL, AX
```

* kan erstattes af:
`LEA BX, ITABEL
MOV AX, [BX+SI]`

```
MOV AX, INDEX
SUB AX, STARTIDX
MOV SI, ELMLEN
MUL SI
MOV SI, AX
* MOV AX, ITABEL+2[SI]
MOV ITAL, AX
```

* kan erstattes af:
`LEA BX, ITABEL
MOV AX, [BX+SI]`

2. dimensional tabel

PASCAL

TYPE

```
DIM1      :ARRAY [1..3]
          OF INTEGER;
```

VAR {main}

```
ITABEL2   :ARRAY [1..5]
          OF DIM1;
IDX1      :INTEGER;
IDX2      :INTEGER;
ITAL      :INTEGER;
```

```
ITABEL2 [IDX2, IDX1] := ITAL;
```

```
ITAL := ITABEL2 [IDX2, IDX1];
```

ASSEMBLER

{datasegment}			
ELMLEN	EQU	2	
STARTIDX1	EQU	1	
STARTIDX2	EQU	1	
ANTDIM1	EQU	3	
ITABEL2	DW	5*3 DUP (?)	

IDX1	DW	?
IDX2	DW	?
ITAL	DW	?

```
MOV AX, ITAL
PUSH AX
MOV AX, IDX2
SUB AX, STARTIDX2
MOV SI, ANTDIM1
MUL SI
ADD AX, IDX1
SUB AX, STARTIDX1
MOV SI, ELMLEN
MUL SI
MOV SI, AX
POP AX
* MOV ITABEL [SI], AX
```

* kan erstattes af:
 LEA BX, ITABEL
 MOV [BX+SI], AX

```
MOV AX, IDX2
SUB AX, STARTIDX2
MOV SI, ANTDIM1
MUL SI
ADD AX, IDX1
SUB AX, STARTIDX1
MOV SI, ELMLEN
MUL SI
MOV SI, AX
* MOV AX, ITABEL [SI]
MOV ITAL, AX
```

* kan erstattes af:
 LEA BX, ITABEL
 MOV AX, [BX+SI]

7.1.d

Implementering af sekventiel tabelgennemløb

Bjørk Busch

Når man skal gennemløbe en tabel sekventielt, kan man vælge at beregne indexadresse for hver tilgang (se foregående afsnit), man kan imidlertid også vælge at optælle adresse istedet for, hvilket giver hurtigere gennemløb.

PASCAL.

VAR

```
TABEL:     ARRAY[1..10] of INTEGER;
TAL:       INTEGER;

FOR I := 1 TO 10 DO BEGIN
    TABEL[I] := 4;
    .....
    TAL := TABEL[I];
END;
```

Omskrevet til assembler:

```
ELMLENGTH EQU 2
TABEL      DW 10 DUP(?)
TAL        DW ?
```

```
FORINIT:  MOV CX,10
          MOV SI,0           ; indexadresse
FORBEG:   MOV AX,4
          MOV [SI],AX
          .....
          MOV AX,[SI]
          MOV TAL,AX
FORNEXT:  ADD SI,ELMLENGTH ; SI -> næste element
          LOOP
```

Udgave uden brug af symbolsk adresse

```
          LEA SI,TABEL ; DS:SI->TABEL[0] før start
          .....
FORINIT:  MOV CX,10
FORBEG:   MOV AX,4
          MOV [SI],AX
          .....
          MOV AX,[SI]
          MOV TAL,AX
FORNEXT:  ADD SI,ELMLENGTH ; SI -> næste element
          LOOP
```

7.2.a

Implementering af procedurer og funktioner

Bjørk Busch

I assembler er der i forbindelse med procedurer og funktioner flere måder at overføre parametre og returværdier på.

Den ene måde er ved at anvende registrene til indhold eller som pointere, en anden måde er ved at anvende stakken.

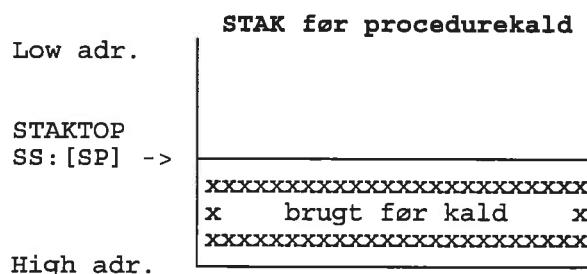
Når man anvender stakken til parameter overførsel kan dette igen gøres på flere måder.

PASCAL anvender stakken på en måde og sproget C gør det en anden. Hovedforskellen er dog primært om det er det kaldende program eller underrutinen, der skal ryde op på stakken.

I PASCAL er det underrutinens opgave at ryde op på stakken og i C er det kaldende program, der selv ryder op.

I det efterfølgende vises hvordan stakken kan anvendes efter den metode, som PASCAL anvender.

Parameteroverførsel ved brug af stakken.



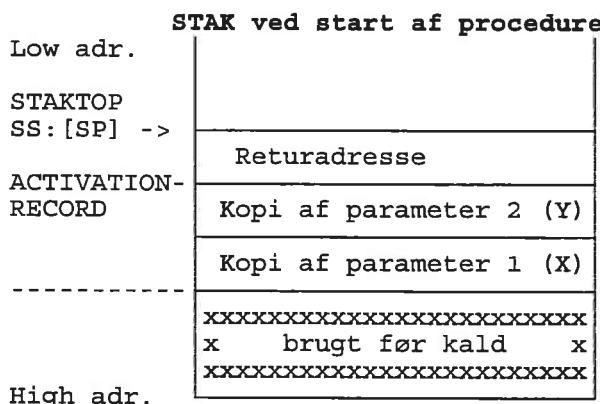
Procedurekald med value parametre.

Proceduren kan være defineret som:

```
PROCEDURE minproc (x,y:integer)
```

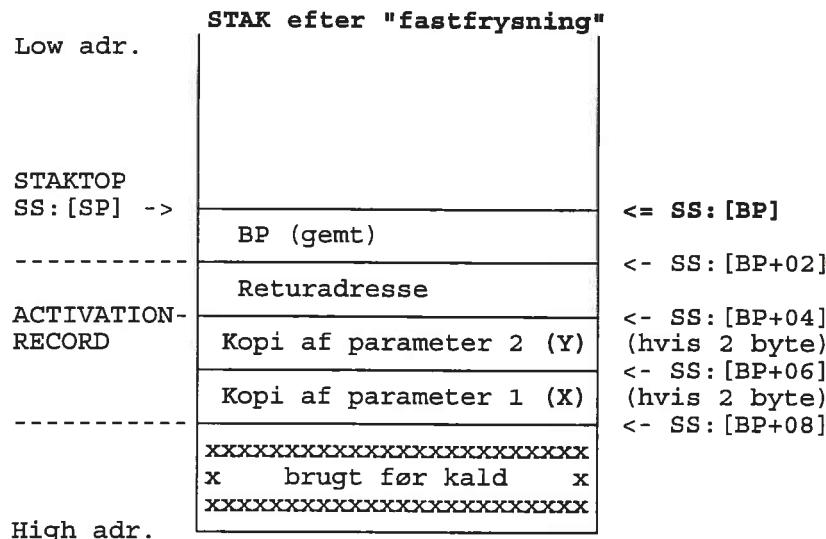
Opstart af proceduren:

```
PUSH    xmain
PUSH    ymain
CALL   minproc
```



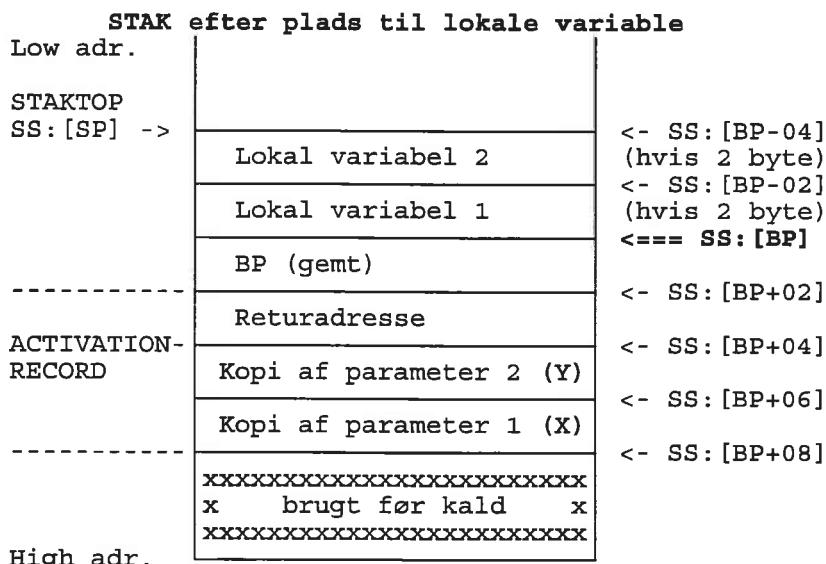
Indledning i procedure (PROLOG) :
 "FASFRYS" udgangspunkt returadr. og parametre

PUSH BP
 MOV BP, SP



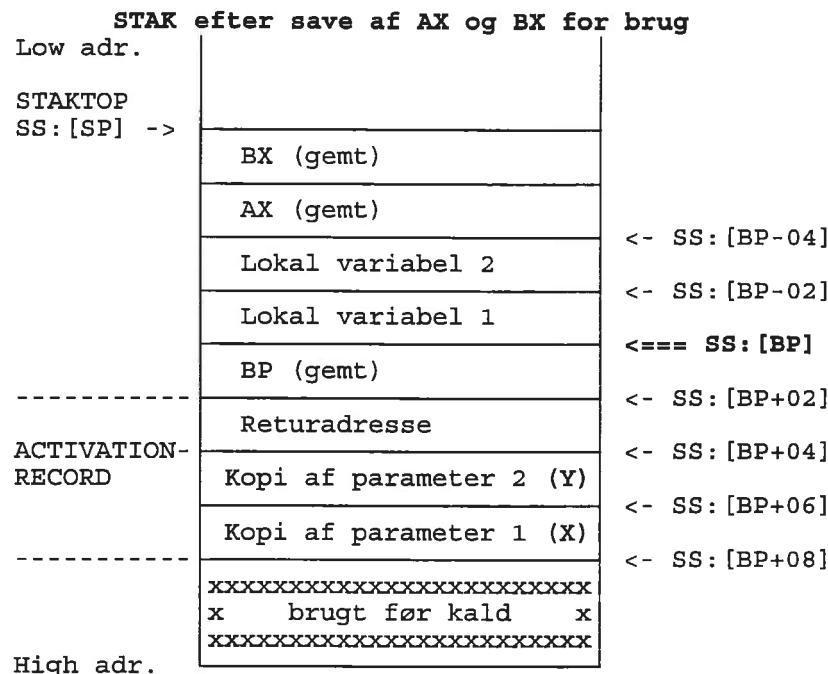
Indledning i procedure fortsat - hvis lokale variable:
 Der skaffes hukommelse til lokale variable (her 2 integer)

SUB Sp, 4



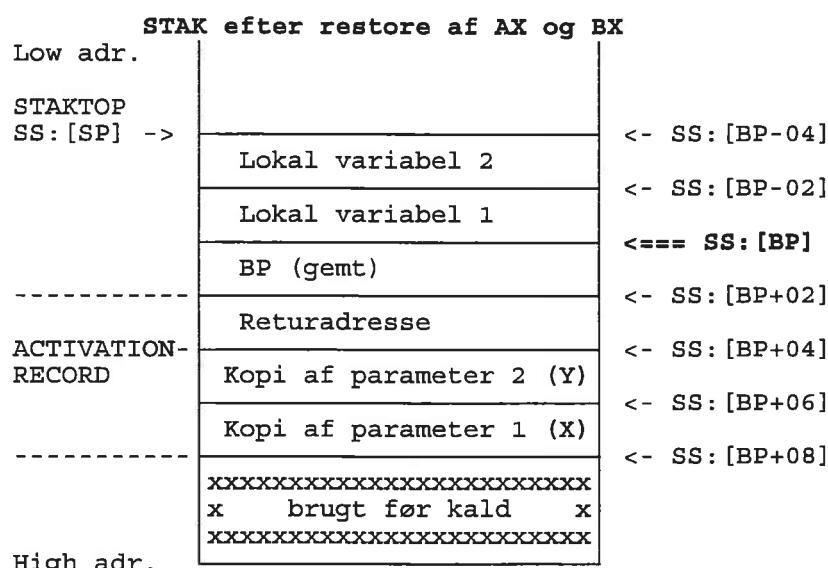
Senere i procedure hvor Arbejdsregistrene AX og BX gemmes:

PUSH AX
PUSH BX



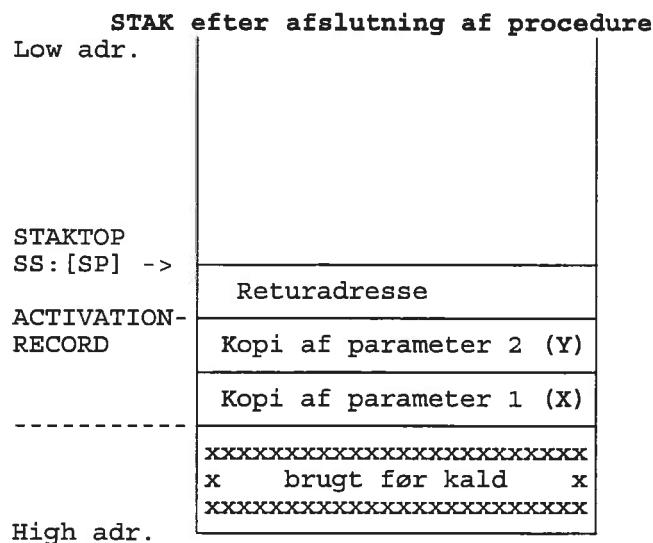
Før afslutning af procedure hvor Arbejdsregistrene AX og BX bliver reetableret:

POP BX
POP AX



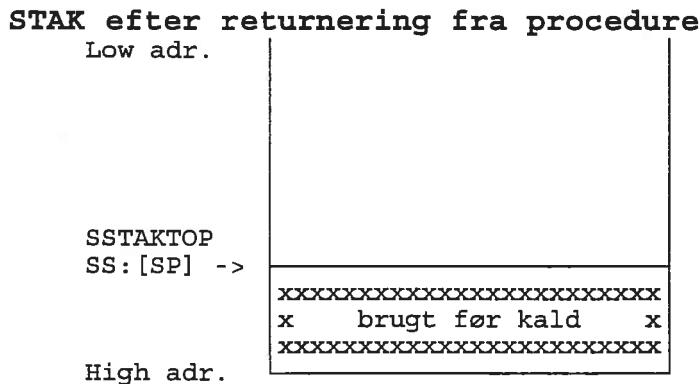
Afslutning af procedure (EPILOG)

```
MOV      SP, BP
POP      BP
```



Returnering fra procedure (parametre fyldte 4 bytes (2 word))

```
RET      4          ;returner og ryd op på stak
```



Procedurer med returparametre.

I forbindelse med procedurer med returparametre overføres **adressen** på parametrene på stakken istedet for indholdet. Procedureindledningen er som tidligere beskrevet, og stakken ser efter denne således ud for en procedure med to VAR-parametre og en lokal variabel af typen integer/word.

Proceduren kan være defineret som:

```
PROCEDURE PROC2 (VAR X,Y: INTEGER)
```

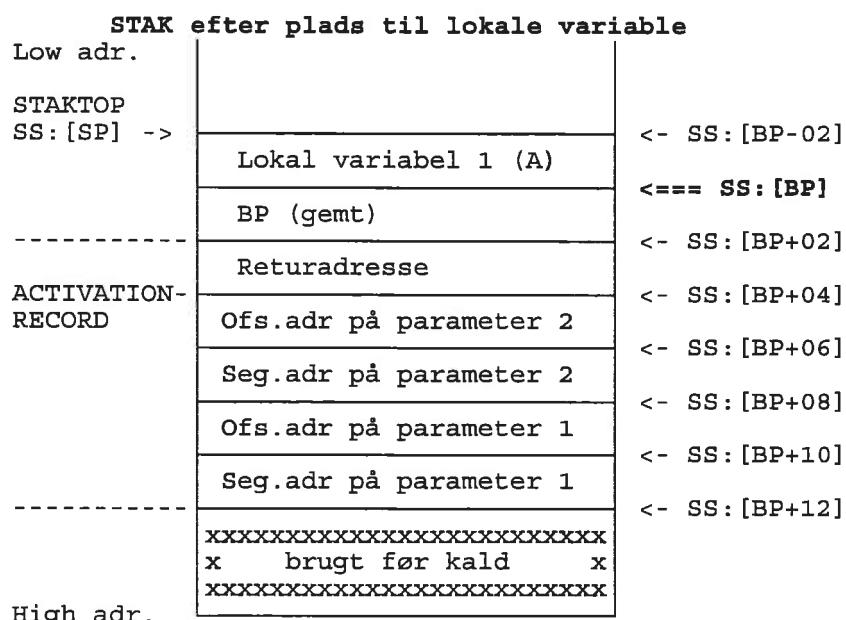
Opstart af proceduren:

LEA	Ax, xmain
PUSH	Ds
PUSH	Ax
LEA	Ax, ymain
PUSH	Ds
PUSH	Ax
CALL	PROC2

Indledning i procedure (PROLOG) :

"FASFRYS" udgangspunkt, returadr., parametre og afsæt plads til lokalvariabel.

PUSH	BP
MOV	BP, SP
SUB	Sp, 2



Hvis proceduren skal ombytte X og Y på følgende måde:

```
A := X;
X := Y;
Y := A;
```

Det centrale i proceduren kan nu ske på følgende måde:

LES	Bx,Ss: [Bp+8]	; Bx:=ofs. og Es:=seg. på X
MOV	Ax,Es: [Bx]	; Ax := X
MOV	Ss: [Bp-2],Ax	; A := Ax
LES	Bx,Ss: [Bp+4]	; Es:Bx := adr. på Y
MOV	Ax,Es: [Bx]	; Ax := Y
LES	Bx,Ss: [Bp+8]	; Es:Bx := adr. på X
MOV	Es: [Bx],Ax	; X := Ax
MOV	Ax, [Bp-2]	; Ax := A
LES	Bx,Ss: [Bp+4]	; Es:Bx := adr. på Y
MOV	Es: [Bx],Ax	; Y := Ax

Bemærk at LES-instruktionen kan erstatte to MOV-instruktioner
Som det sikker fremgår, er det muligt at spare nogle
instruktioner, hvis man bruger flere arbejdsregistre.

Afslutning af procedure (EPILOG)

```
MOV      SP,BP
POP      BP
```

Returnering fra procedure (adresse-parametre fyldte 8 bytes
(hver adresse var på 2*2 word))

```
RET      8           ;returner og ryd op på stak
```

Eksempel på hvordan procedure GETDATE kan se ud.

Proceduren er i pascal defineret på følgende måde:

Procedure GetDate (var aa:integer; var mm, dd, ugedg: byte);

```

GetDate Proc Near
    PUSH Bp ;gem basispointer
    MOV Bp, Sp ;Bp -> staktop
                ; sidste parameter er 4 højere
    PUSH Ax ;gem arbejdsregistre
    PUSH Bx
    PUSH Cx
    PUSH Dx
    PUSH Es

    MOV Ah, 2Ah ;funktion GET DATE
    INT 21h ;Dos service funktion

    LES Bx, Ss: [Bp+4] ;Es:Bx := adr. på sidste parm
    MOV Es: [Bx], Al ;returner ugedag

    LES Bx, Ss: [Bp+8] ;Es:Bx := adr. på 2. sidste parm
    MOV Es: [Bx], Dl ;returner dag

    LES Bx, Ss: [Bp+12] ;Es:Bx := adr. på 3. sidste parm
    MOV Es: [Bx], Dh ;returner måned

    LES Bx, Ss: [Bp+16] ;Es:Bx := adr. på 4. sidste parm
    MOV Es: [Bx], Cx ;returner årstal

    POP Es ;restore arbejdsregistre
    POP Dx
    POP Cx
    POP Bx
    POP Ax

    POP Bp ; restore basispointer
    RET 16 ;returner og juster stak (parm 4*(2 ord))
GetDate EndP

```

Eksempel på brug af proceduren:

Variable i datasegmentet, som antages at være sat op i DS.

```

.....
AARSTAL DW ?
MAANED DB ?
DAG DB ?
UGEDAG DB ?

.....
LEA AX, AARSTAL
PUSH DS ; 1. parameters adresse
PUSH AX ; lægges på stak
LEA AX, MAANED
PUSH DS ; 2. parameters adresse
PUSH AX ; lægges på stak
LEA AX, DAG
PUSH DS ; 3. parameters adresse
PUSH AX ; lægges på stak
LEA AX, UGEDAG
PUSH DS ; 4. parameters adresse
PUSH AX ; lægges på stak
CALL GetDate
.....

```

Funktioner:

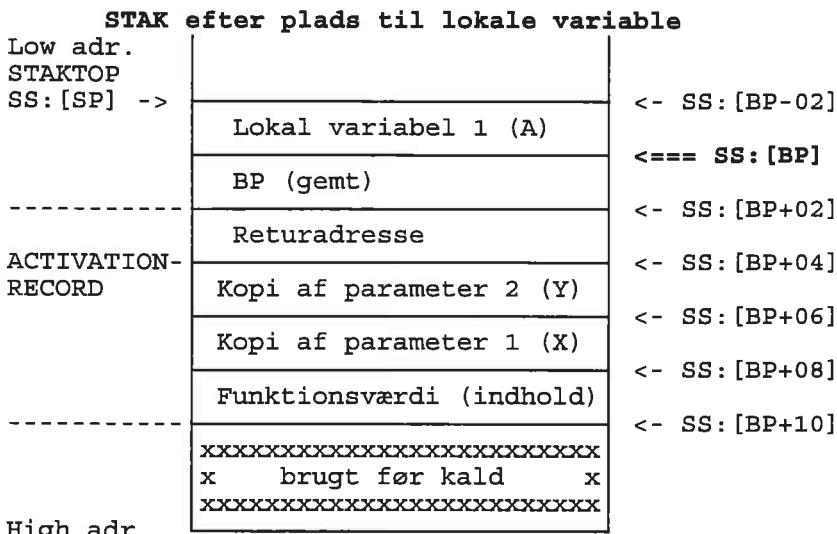
I forbindelse med **Funktioner** returneres **funktionsværdien** på stakken.

Det kaldende program afsætter plads til funktionsværdien.

Bliver funktionen kaldt med parametre, overføres disse på samme måde som der gælder for procedurer, idet der først sættes plads til funktionsværdien og herefter udlægges parametre.

Funktionen kan være defineret som:

```
FUNCTION SUM (X, Y: INTEGER) : INTEGER
```



Hvis funktionen skal returnere summen af de 2 parametre kan den se ud på følgende måde:

```
SUM      Proc    Near
PUSH    Bp          ; gem basispointer
MOV     Bp, Sp       ; Bp -> staktop
                  ; sidste parameter er 4 højere
PUSH    Ax          ; gem arbejdsregistre
MOV     Ax, SS:[Bp+6] ; AX := X
ADD    Ax, SS:[Bp+4] ; AX := AX + Y
MOV     SS:[Bp+8], Ax ; funktionsværdi := AX
POP    Bp          ; restore basispointer
RET    4           ; returner og juster stak, idet
                  ; funktionsværdi efterlades på stak-top
SUM      EndP
```

Eksempel på brug af funktionen:

Variable i databasegmentet, som antages at være sat op i DS.

```
TAL1    DW      3
TAL2    DW      4
SUMTAL DW      ?

SUB    SP, 2       ; Afsæt plads til funktionsværdi
PUSH   TAL1        ; overfør parameter 1 til stak (kopi)
PUSH   TAL2        ; overfør parameter 2 til stak (kopi)
CALL   SUM         ; kør funktionsproceduren
POP    SUMTAL      ; SUMTAL := funktionsværdi
```

Bemærk at PUSH og POP kun kan anvendes ved word-parametre, ellers må man selv justere SP-registeret og bruge MOV-instruktionen til at flytte data med.

Flere eksempler på procedurer og funktioner.

PROCEDURE READCHAR (VAR TEGN: CHAR);

Hent tegn fra tastaturlbuffer.
Procedure kodet i assembler.

```
ReadChar Proc Near
    PUSH Bp           ; gem basispointer
    MOV  Bp, Sp        ; Bp -> staktop
                    ; sidste parameter er 4 højere
    PUSH Ax           ; gem arbejdsregistre
    PUSH Bx
    PUSH Es
    MOV  Ah, 08h       ; funktion INPUT - ECHO + CTRL C
    INT  21h
    LES  Bx, Ss: [Bp+4] ; Es:Bx := adr. på sidste parm
    MOV  Es: [Bx], Al   ; returner tegn
    POP  Es
    POP  Bx
    POP  Ax           ; restore arbejdsregistre

    POP  Bp           ; restore basispointer
    RET  4             ; returner og juster stak
ReadChar EndP
```

Anvendelse fra assembler.

```
TEGN    DB    ?
.....
    LEA    AX, TEGN
    PUSH DS           ; 1. parameters adresse
    PUSH AX           ; lægges på stak
    CALL ReadChar
```

Samme rutine men som funktion:

FUNCTION READKEY : CHAR;

Procedure kodet i assembler.

```
ReadKey Proc Near
    PUSH Bp           ; gem basispointer
    MOV  Bp, Sp        ; Bp -> staktop
                    ; sidste parameter er 4 højere
    PUSH Ax           ; gem arbejdsregistre
    MOV  Ah, 08h       ; funktion INPUT - ECHO + CTRL C
    INT  21h
    MOV  Ss: [Bp+4], Al ; Returner funktionsværdi
    POP  Ax

    POP  Bp           ; restore basispointer
    RET  0             ; returner og juster stak
ReadKey EndP
```

Anvendelse fra assembler.

```
TEGN    DB    ?
.....
    SUB  Sp, 1          ; Sæt plads til funktionsværdi
    CALL ReadKey
    MOV  Bp, Sp         ; Hent funktionsværdi
    MOV  Al, Ss: [Bp]    ; fra stakken
    ADD  Sp, 1          ; Fjern fra stak
    MOV  TEGN, Al        ; Overfør funktionsværdi
```

Som det kan ses er det noget besværlig når der arbejdes med ulige antal bytes på stak. Der kan anvendes word istedet, hvor den ene byte så blot ignoreres.

```
PROCEDURE WRITECHR (TEGN: CHAR);
```

Skriv tegn på skærm.
procedure kodet i assembler.

```
WriteChr Proc Near
    PUSH Bp ;gem basispointer
    MOV Bp, Sp ;Bp -> staktop
            ; sidste parameter er 4 højere
    PUSH Ax ;gem arbejdsregistre
    PUSH Dx
    MOV Dl, Ss: [Bp+4] ;Dl := parameterværdi
    MOV Ah, 02h ;funktion OUTPUT CHARACTER
    INT 21h ;Dos service funktion
    POP Dx ;restore arbejdsregistre
    POP Ax

    POP Bp ;restore basispointer
    RET 1 ;returner og juster stak
WriteChr EndP
```

Opstart fra assembler.

```
TEGN DB ?
.....
    SUB Sp, 1 ; Plads til parameter
    MOV Bp, Sp ; Overfør parameter
    MOV Al, Tegn ; til stak
    MOV Ss: [Bp], Al ;(ulige antal byte er besværlig)
    CALL WriteChr
```

Som det kan ses er det noget besværlig når der arbejdes med ulige antal bytes på stak. Der kan anvendes word istedet, hvor den ene byte så blot ignoreres.

Hvis der anvedes word istedet ville procedure og kald se således ud:

```
WriteChr Proc Near
    PUSH Bp ;gem basispointer
    MOV Bp, Sp ;Bp -> staktop
            ; sidste parameter er 4 højere
    PUSH Ax ;gem arbejdsregistre
    PUSH Dx
    MOV Dl, SS: [Bp+4] ;Dl := parameterværdi
    MOV Ah, 02h ;funktion OUTPUT CHARACTER
    INT 21h ;Dos service funktion
    POP Dx ;restore arbejdsregistre
    POP Ax

    POP Bp ;restore basispointer
    RET 2 ;returner og juster stak
WriteChr EndP
```

Opstart fra assembler.

```
TEGN DB ?
.....
    PUSH TEGN ;Overfør parameter til stak
    CALL WriteChr
```

Det blev det noget enklere af ikke !!!

Bemærk at proceduren skal tage 1 mere af stakken.

I praksis anvendes normalt 2 byte, da det er enklere.

```
FUNCTION GETCHARXY (X, Y: Integer);
```

Hent tegn fra skærbuffer position X,Y (hjørne = 1,1).
Procedure kodet i assembler.

```
SKRMSEG EQU 0B800h ;Kunne også findes via. skærmmodus
GetCharXY Proc Near
    PUSH Bp ;gem basispointer
    MOV Bp, Sp ;Bp -> staktop
    ; sidste parameter er 4 højere
    PUSH Ax ;gem arbejdsregistre
    PUSH Si
    PUSH Es
    ;
    MOV Ax,Ss: [Bp+4] ;Ax := Y
    DEC Ax ;ryk start til 0
    MOV Si,80 ;tegn pr. linie
    MUL Si ;Si := start tegn for linie
    ADD Ax,Ss: [Bp+6] ;adder kolonne = X
    DEC Ax ;ryk X-start til 0
    SHL Ax,1 ;Ax := tegnoffset 2 byte pr. tegn
    MOV Si,Ax ;Si := udvalgt tegnadresse
    MOV Ax,SKRMSEG
    MOV Es,Ax ;opsæt skærbuffer segmentadr.
    MOV Al,Es:[Si] ;hent ascii fra skærbuffer
    XOR Ah,Ah ;udvid til word - nulstil Ah
    MOV Ss:[Bp+8],Ax ;overfør funktionsværdi til stak
    POP Es
    POP Si
    POP Ax
    POP Bp ;restore basispointer
    RET 4 ;returner og juster stak
    ;funktionsværdi bliver på stak
GetCharXY EndP
```

Opstart fra assembler.

```
TEGN DB ?
X DW 4
Y DW 12
.....
SUB Sp,2 ;plads til funktionsværdi
PUSH X ;1. parameters indhold på stak
PUSH Y ;2. parameters indhold på stak
CALL GetCharXY
POP Ax ;Hent funktionsværdi og
MOV Tegn,Al ;overfør den til Tegn.
```

Det skal bemærkes, at der her er anvendt word parametre i procedure og "afkortning" først sker efter kaldet ved overførselen af funktionsværdien til TEGN.

Det er muligt at gøre procedurerne mere "læsevenlige" ved brug af EQU sætninger. Dette vises på næste side med samme procedure.

FUNCTION GETCHARXY (X, Y: Integer);

Denne gang mere "læsevenlig" ved brug af EQU-sætninger.

```

SKRMSEG EQU 0B800h
Y EQU WORD PTR Ss:[Bp+4] ; 2. parameter
X EQU WORD PTR Ss:[Bp+6] ; 1. parameter
FUNK EQU WORD PTR Ss:[Bp+8] ; funktionsværdi - retur

GetCharXY Proc Near
    PUSH Bp           ; gem basispointer
    MOV  Bp, Sp        ; Bp -> staktop
                      ; sidste parameter er 4 højere
    PUSH Ax           ; gem arbejdsregistre
    PUSH Di
    PUSH Es

    MOV  Ax,Y          ; Ax := Y
    DEC Ax             ; ryk start til 0
    MOV  Di,80          ; tegn pr. linie
    MUL  Di             ; Di := start tegn for linie
    ADD  Ax,X           ; adder X
    DEC  Ax             ; ryk start til 0
    SHL  Ax,1            ; Ax := tegnoffset 2 byte pr. tegn
    MOV  Di,Ax          ; Di := udvalgt tegnadresse
    MOV  Ax,SKRMSEG
    MOV  Es,Ax          ; opsæt skærmbuffer segmentadr.
    MOV  Al,Es:[Di]      ; hent ascii fra skærmbuffer
    XOR  Ah,Ah
    MOV  FUNK,Ax          ; overfør funktionsværdi til stak

    POP  Es
    POP  Di
    POP  Ax

    POP  Bp           ; restore basispointer
    RET  4             ; returner og juster stak
                      ; funktionsværdi bliver på stak
GetCharXY EndP

```

Under oversættelsen erstattes symbolnavne fra EQU med det de er defineret som og der er derfor ingen forskel på den oversatte procedure i forhold til foregående udgave.

Prøv selv at lave følgende procedurer/funktioner med tilhørende eksempel på kald:

Du kan også prøve at afteste under CODEVIEW.

Til løsning kan anvendes BIOS-kald 10h.

- 1) PROCEDURE GOTOXY (X, Y: Integer)
Placer cursor
- 2) FUNCTION GETX : Integer; og FUNCTION GETY : Integer;
Returner cursor position, henholdsvis X og Y koordinat.
- 3) PROCEDURE GETXY (VAR X, Y: Integer);
Returner både X og Y koordinat

7.5.a

Programeksempel i pascal

Bjørk Busch

Dette programeksempel er udgangspunkt for senere implementering med inline assembler og implementering i "ren" assembler.

```

{$R-} {$S-} {$I-}      {compiler-direktiver -range -stak -io check}
{$V-}           {ingen strengtypecheck}

Program SumSnit;
Type
  StalType = String[5];
  ItabType = Array[1..100] of Integer;

Procedure Bin2Asc(Ital: Integer; Var Stal: StalType);
Var
  I: Integer;
  Wtal: Integer;
begin
  Stal[0] := Chr(5);
  For I := 5 downto 1 do begin
    Wtal := Ital MOD 10;
    Stal[I] := Chr(Wtal+Ord('0'));
    Ital := Ital DIV 10;
  end;
end;

Procedure WriteStr(Var Streng: String);
Var
  I : Integer;
  Antal: Integer;
begin
  Antal := Length(Streng);
  I := 1;
  While Antal > 0 do begin
    Write(Streng[I]);
    I := I + 1;
    Antal := Antal - 1;
  end;
end;

Procedure WriteInt(Ital: Integer);
Var
  Stal: StalType;
begin
  Bin2Asc(Ital,Stal);
  WriteStr(Stal);
  WriteLN;
end;

Procedure WriteIntTab(Antal: Integer; Var Itab: ItabType);
Var
  I: Integer;
  Ital: Integer;
begin
  I := 1;
  While Antal > 0 do begin
    Ital := Itab[I];
    WriteInt(Ital);
    I := I + 1;
    Antal := Antal - 1;
  end;
end;

```

```

Procedure Asc2Bin(Var Stal: StalType; Var Ital: Integer);
Var
  I:      Integer;
  L:      Integer;
  Wtall1: Integer;
  Wtall2: Integer;
begin
  L := Ord(Stal[0]);
  Wtall1 := 0;
  For I := 1 to L do begin
    Wtall1 := Wtall1 * 10;
    Wtall2 := Ord(Stal[I]) - Ord('0');
    Wtall1 := Wtall1 + Wtall2;
  end;
  Ital := Wtall1;
end;

Procedure ReadInt(Var Tekst: String; Var Ital: Integer);
Var
  Stal: StalType;
  Wtal: Integer;
begin
  WriteStr(Tekst);
  ReadLN(Stal);
  Asc2Bin(Stal,Wtal);
  Ital := Wtal;
end;

Procedure ReadIntTab(Var Tekst: String; Antal: Integer;
                     Var Itab: ItabType);
Var
  I:      Integer;
  Ital: Integer;
begin
  I := 1;
  While Antal > 0 do begin
    ReadInt(Tekst,Ital);
    Itab[I] := Ital;
    I := I + 1;
    Antal := Antal - 1;
  end;
end;

Procedure IntTabSum(Antal: Integer; Var Itab: ItabType;
                    Var Sum: Integer);
Var
  I:      Integer;
  Wsum: Integer;
begin
  I := 1;
  Wsum := 0;
  While Antal > 0 do begin
    Wsum := Wsum + Itab[I];
    I := I + 1;
    Antal := Antal - 1;
  end;
  Sum := Wsum;
end;
Procedure WriteIntTabSum(Antal: Integer; Var Itab: ItabType);
Var
  Sum: Integer;
begin
  IntTabSum(Antal,Itab,Sum);
  WriteInt(Sum);
end;
Procedure IntTabGsnit(Antal: Integer; Var Itab: ItabType;
                      Var Gsnit: Integer);
Var
  Wsum: Integer;
begin
  IntTabSum(Antal,Itab,Wsum);
  Gsnit := Wsum DIV Antal;
end;

```

```
Procedure WriteIntTabGsnit(Antal: Integer; Var Itab: ItabType);
Var
  Gsnit: Integer;
begin
  IntTabGsnit(Antal, Itab, Gsnit);
  WriteInt(Gsnit);
end;
```

{Main}

```
Var
  Tekst: String;
  Itab: ItabType;
  Antal: Integer;
  Sum: Integer;
  Gsnit: Integer;

begin
  Tekst := 'Beregning af sum og gennemsnit'#13#10;
  WriteStr(Tekst);

  Tekst := 'Intast positivt heltal: ';
  Antal := 5;
  ReadIntTab(Tekst, Antal, Itab);

  Tekst := 'Oversigt over tallene:'#13#10;
  WriteStr(Tekst);

  WriteIntTab(Antal, Itab);

  Tekst := 'Summen er: ';
  WriteStr(Tekst);

  WriteIntTabSum(Antal, Itab);

  Tekst := 'Gennemsnittet er: ';
  WriteStr(Tekst);

  WriteIntTabGsnit(Antal, Itab);

  ReadLN;
end.
```

7.5.b

Programeksempel i pascal med inline assembler

Bjørk Busch

Dette programeksempel bygger på det foregående pascalprogram, men implementeringen er i nedenstående eksempel foretaget med inline assembler.

Der er i denne oversættelse ikke foretaget nogen form for optimering, idet hver pascal-sætning er oversat for sig.

Turbo-Pascal forventer kun at registrene Ds, Ss, Sp og Bp intakte, og i eksemplet er det derfor kun disse registre, der reestablisheres i forbindelse med procedurerne.

I forbindelse med indlæsning med READLN har det været nødvendigt, at definere en indlæsningsbuffer, der er anvendt istedet for den oprindelige streng "Stal", og der er ikke foretaget kopiering af data til denne oprindelige streng.

I main-rutinen er tilskrivninger af faste tekster til streng-variable ikke oversat. Disse tilskrivninger vil kræve en flytning af hvert eneste karakter for sig.

```

{$R-} {$S-} {$I-}      {compiler-direktiver -range -stak -io check }
{$V-}                  {ingen strengtypecheck}
Program SumSnit;

Type
  StalType = String[5];
  ItabType = Array[1..100] of Integer;

Procedure Bin2Asc(Ital: Integer; Var Stal: StalType);
Var
  I: Integer;
  Wtal: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
    Push      Bp
    Mov       Bp,Sp
    Sub      Sp,4           plads til lokal-var

  Der er desuden oprettet følgende symboler:
  Wtal     Equ      [Bp-4]
  I        Equ      [Bp-2]
  Stal    Equ      [Bp+4]     Adresse
  Ital     Equ      [Bp+8]      Value
-----}
  ASM
    LES      Bx,Ss:[Stal]      {Stal[0] := Chr(5); }
    Mov      Al,5
    Mov      Es:[Bx],Al
{ForInit}
    Mov      Ax,5
    Mov      Ss:[I],Ax          {For I := 5 downto 1 do begin}
@ForTst:
    Mov      Ax,Ss:[I]
    Cmp      Ax,1
    Jl       @EndFor
{ForSave}
    Push     Ss:[I]
{ForBody}
    Mov      Bx,10            {Wtal := Ital MOD 10; }
    Mov      Ax,Ss:[Ital]
    Cwd
    Div      Bx
    Mov      Ss:[Wtal],Dx      {; Ax=Kvotient, Dx=Rest }

    Mov      Ax,Ss:[Wtal]
    Add      Ax,'0'
    Mov      Di,Ss:[I]
    LES      Bx,Ss:[Stal]
    Mov      Es:[Bx+Di],Al      {Stal[I] := Chr(Wtal+ Ord('0')); }

    ; udregn adr. på Stal[I]
    ; NB! elm.længde=1 byte
    ; NB! start=0

    Mov      Bx,10            {Ital := Ital DIV 10; }
    Mov      Ax,Ss:[Ital]
    Cwd
    Div      Bx
    Mov      Ss:[Ital],Ax      {; Ax=Kvotient, Dx=Rest }

{ForRestore}
    Pop      Ss:[I]           {End}
{ForNext}
    Dec      Word PTR Ss:[I]
    Jmp      @ForTst
@EndFor:
    END;
{-----
  pascal har her selv indskudt følgende epilog
    Mov      Sp,Bp           retabler stak
    Pop      Bp
    Ret      6                2+4      fjern activationrecord
-----}
end;

```

```

Procedure WriteStr(Var Streng: String);
Var
  I      : Integer;
  Antal: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
  Push    Bp
  Mov     Bp,Sp
  Sub     Sp,4           plads til lokal-var

Der er desuden oprettet følgende symboler:
Antal   Equ      [Bp-4]
I       Equ      [Bp-2]
Streng  Equ      [Bp+4]           adresse
-----}
ASM
  LEs     Bx,Ss:[Streng]      { Antal:=Length(Streng); }
  Mov     Al,Es:[Bx]          { ; Ax := Al; }
  Cbw
  Mov     Ss:[Antal],Ax

  Mov     Word PTR Ss:[I],1    { I := 1; }

@WhileTst:
  Cmp     Word PTR Ss:[Antal],0 { While Antal > 0 }
  Jle     @EndWhile

@WhileBody:
  Mov     Si,Ss:[I]
  LEs     Bx,Ss:[Streng]
  Mov     Dl,Es:[Bx+Si]        { do begin
  Mov     Ah,02                 Write(Streng[I]);
  Int     21h                  ; udregn adr.på Stal[I]
                                ; NB! elm.længde=1 byte
                                ; NB! start=0
                                { end; }

  Inc     Word PTR Ss:[I]      { I := I + 1;
  Dec     Word PTR Ss:[Antal]  Antal := Antal - 1; }

  Jmp     @WhileTst

@EndWhile:
  END;
{-----
  pascal har her selv indskudt følgende epilog
  Mov     Sp,Bp               retabler stak
  Pop    Bp
  Ret     4                   fjern activationrecord
-----}
end;

```

```

Procedure WriteInt(Ital: Integer);
Var
  Stal: StalType;
begin
{-----
  pascal har her selv indskudt følgende prolog
  Push      Bp
  Mov       Bp, Sp
  Sub       Sp, 6           plads til lokal-var

  Der er desuden oprettet følgende symboler:
  Stal     Equ      [Bp-6]
  Ital     Equ      [Bp+4]      value
-----}
ASM
  Mov      Ax, [Ital]      { Bin2Asc(Ital,Stal);  }
  Push     Ax
  Lea      Bx, [Stal]
  Push     Ss               { adr. på Stal }
  Push     Bx
  Call    Bin2Asc

  Lea      Bx, [Stal]      { WriteStr(Stal);  }
  Push     Ss               { adr. på Stal }
  Push     Bx
  Call    WriteStr

  Mov      Ah, 2            { WriteLN }
  Mov      Dl, 13
  Int     21h
  Mov      Dl, 10
  Int     21h
END;
{-----
  pascal har her selv indskudt følgende epilog
  Mov      Sp, Bp          retabler stak
  Pop     Bp
  Ret     2                 fjern activationrecord
-----}
end;

```

```

Procedure WriteIntTab(Antal: Integer; Var Itab: ItabType);
Var
  I: Integer;
  Ital: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
  Push    Bp
  Mov     Bp,Sp
  Sub     Sp,4           plads til lokal-var

  Der er desuden oprettet følgende symboler:
  Ital    Equ      [Bp-4]
  I       Equ      [Bp-2]
  Itab   Equ      [Bp+4]     Adresse
  Antal  Equ      [Bp+8]      Value
-----}
  ASM
@WhileTst:
  Cmp    Word PTR Ss:[Antal],0  { While Antal > 0 }
  Jle    @EndWhile

@WhileBody:
  Mov    Ax,Ss:[I]          { do begin
  Sub    Ax,1               Ital := Itab[I];
  Mov    Bx,2               juster til startindex 0
  Mul    Bx               { elementlængde = 2
  Mov    Si,Ax
  LES   Bx,Ss:[Itab]
  Mov    Ax,Es:[Bx+Si]
  Mov    Ss:[Ital],Ax
  Beregn indexadr.

  Push   Ss:[Ital]          { WriteInt(Ital); }
  Call   WriteInt

  Inc    Ss:[I]              { I := I + 1; }

  Dec    Ss:[Antal]          { Antal := Antal - 1; }

  Jmp    @WhileTst          { end; }

@EndWhile:
  END;
{-----
  pascal har her selv indskudt følgende epilog
  Mov    Sp,Bp             retabler stak
  Pop   Bp
  Ret    6                  fjern activationrecord
-----}
end;

```

```

Procedure Asc2Bin(Var Stal: StalType; Var Ital: Integer);
Var
  I:      Integer;
  L:      Integer;
  Wtal1: Integer;
  Wtal2: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
    Push      Bp
    Mov       Bp, Sp
    Sub      Sp, 8           plads til lokal-var

Der er desuden oprettet følgende symboler:
Wtal2   Equ      [Bp-8]
Wtal1   Equ      [Bp-6]
L        Equ      [Bp-4]
I        Equ      [Bp-2]
Ital    Equ      [Bp+4]      adresse
Stal    Equ      [Bp+8]      adresse
-----}
ASM
  LES      Bx,Ss:[Stal]      { L := Ord(Stal[0]); }
  Mov     Al,Es:[Bx]
  Cbw
  Mov     Ss:[L],Ax
  Mov     Word PTR Ss:[Wtal1],0  { Wtal1 := 0; }

{ForInit}
  Mov     Ax,1
  Mov     Ss:[I],Ax          { For I := 1 to L do begin }
@ForTst:
  Mov     Ax,Ss:[I]
  Cmp     Ax,L
  Jg      @EndFor
{ForSave}
  Push    Ss:[I]
{ForBody}
  Mov     Ax,Ss:[Wtal1]      { Wtal1 := Wtal1 * 10; }
  Mov     Bx,10
  Mul     Bx
  Mov     Ss:[Wtal1],Ax
  Mov     Si,Ss:[I]          { Wtal2 := Ord(Stal[I]) - }
  LES      Bx,Ss:[Stal]
  Mov     Al,Es:[Bx+Si]      { NB! startadr. = 0 }
  Sub     Al,'0'            { Ord('0'); }
  Cbw
  Mov     Ss:[Wtal2],Ax
  Mov     Ax,Ss:[Wtal2]      { Wtal1 := Wtal1 + Wtal2; }
  Add     Ax,Ss:[Wtal1]
  Mov     Ss:[Wtal1],Ax
{ForRestore}
  Pop     Ss:[I]             { End; }
{ForNext}
  Inc     Word PTR Ss:[I]
  Jmp     @ForTst
@EndFor:
  Mov     Ax,Ss:[Wtal1]      { Ital := Wtal1; }
  LES      Di,Ss:[Ital]
  Mov     Es:[Di],Ax
END;
{-----
  pascal har her selv indskudt følgende epilog
    Mov      Sp,Bp           retabler stak
    Pop      Bp
    Ret     8      4+4       fjern activationrecord
-----}
end;

```

```

Procedure ReadInt(Var Tekst: String; Var Ital: Integer);
Type
  BufferType = RECORD
    BufLen: Byte;
    BufArr: String[6];
  END;
Var
  Stal: StalType;
  Wtal: Integer;
  Buf: BufferType;           { bruges istedet for oprindelig Stal}
begin
{-----
  pascal har her selv indskudt følgende prolog
  Push      Bp
  Mov       Bp, Sp
  Sub      Sp, 16          plads til lokal-var

Der er desuden oprettet følgende symboler:
Buf        Equ      [Bp-16]
Buf.BufLen Equ      [Bp-16]
Buf.BufArr Equ      [Bp-15]
Wtal       Equ      [Bp-8]
Stal        Equ      [Bp-6]
Ital        Equ      [Bp+4]      adresse
Tekst       Equ      [Bp+8]      adresse
-----}

ASM
  LES      Bx, [Tekst]      { WriteStr(Tekst);   }
  Push    Es                { adr. på Tekst }
  Push    Bx
  Call   WriteStr

  Push    Ds                { ReadLN(Stal)  }
  Mov     Ax, Ss
  Mov     Ds, Ax
  Mov     Al, 6
  Mov     Ss: [Buf.Buflen], Al
  Lea     Dx, [Buf]
  Mov     Ah, 0Ah
  Int    21h
  Mov     Dl, 10
  Mov     Ah, 2
  Int    21h
  Pop    Ds

  Lea     Bx, [Buf.Bufarr]  { Asc2Bin(Buffarr,Wtal);   }
  Push    Ss                { adr. på lokalvar. BufArr}
  Push    Bx
  Lea     Bx, [Wtal]        { adr. på lokalvar Wtal }
  Push    Ss
  Push    Bx
  Call   Asc2Bin

  Mov     Ax, Ss: [Wtal]    { Ital := Wtal }
  LES    Bx, [Ital]
  Mov     Es: [Bx], Ax

END;
{-----
  pascal har her selv indskudt følgende epilog
  Mov     Sp, Bp          retabler stak
  Pop    Bp
  Ret    8                 4+4      fjern activationrecord
-----}
end;

```

```

Procedure ReadIntTab(Var Tekst: String; Antal: Integer;
                     Var Itab: ItabType);
Var
  I:     Integer;
  Ital: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
  Push    Bp
  Mov     Bp,Sp
  Sub     Sp,4           plads til lokal-var

Der er desuden oprettet følgende symboler:
Ital    Equ      [Bp-4]
I       Equ      [Bp-2]
Itab   Equ      [Bp+4]      adresse
Antal  Equ      [Bp+8]      value
Tekst  Equ      [Bp+10]     adresse
-----}
ASM
@WhileTst:
  Cmp
  Jle  @EndWhile
{While-body}
  LES   Bx, [Tekst]        { ReadInt(Tekst, Ital); }
  Push  Es
  Push  Bx
  Lea   Bx, [Ital]
  Push  Ss
  Push  Bx
  Call  ReadInt

  Mov   Ax,Ss:[Ital]       { Itab[I] := Ital; }
  Push  Ax
  Mov   Ax,Ss:[I]          { ;gem italic
                           ;beregn indexadr. }
  Sub   Ax,1
  Mov   Bx,2
  Mul   Bx
  Mov   Di,Ax
  LES   Bx,Ss:[Itab]
  Pop   Ax
  Mov   Es:[Bx+Di],Ax      { ;hent italic }

  Inc   Word PTR Ss:[I]    { I := I + 1; }
  Dec   Word PTR Ss:[Antal] { Antal := Antal - 1; }

  Jmp   @WhileTst         { end; }

@EndWhile:
  END;
{-----
  pascal har her selv indskudt følgende epilog
  Mov   Sp,Bp             retabler stak
  Pop   Bp
  Ret   10     4+2+4      fjern activationrecord
-----}
end;

```

```

Procedure IntTabSum(Antal: Integer; Var Itab: ItabType;
                     Var Sum: Integer);
Var
  I:     Integer;
  Wsum: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
    Push   Bp
    Mov    Bp, Sp
    Sub    Sp, 4           plads til lokal-var

Der er desuden oprettet følgende symboler:
Wsum      Equ      [Bp-4]
I         Equ      [Bp-2]
Sum       Equ      [Bp+4]      adresse
Itab      Equ      [Bp+8]      adresse
Antal     Equ      [Bp+12]     Value
-----}
ASM
  Mov      Word PTR Ss:[I],1    { I := 1; }
  Mov      Word PTR Ss:[Wsum],0  { Wsum := 0; }

@WhileTst:
  Cmp      Word PTR Ss:[Antal],0
  Jle      @EndWhile
{While-body}
  Mov      Ax,Ss:[I]          { Wsum := Wsum+Itab[I]; }
  Sub      Ax,1               { juster til startindex 0 }
  Mov      Bx,2               { elementlængde = 2 }
  Mul      Bx                 { beregn indexadr. }
  Mov      Si,Ax
  LEs      Bx,Ss:[Itab]
  Mov      Ax,Es:[Bx+Si]
  Add      Ax,Ss:[Wsum]
  Mov      Ss:[Wsum],Ax

  Inc      Word PTR Ss:[I]    { I := I + 1; }
  Dec      Word PTR Ss:[Antal] { Antal := Antal - 1; }

  Jmp      @WhileTst        { end; }
@EndWhile:
  Mov      Ax,Ss:[Wsum]       { Sum := Wsum; }
  LEs      Bx,Ss:[Sum]
  Mov      Es:[Bx],Ax

END;
{-----
  pascal har her selv indskudt følgende epilog
    Mov    Sp,Bp             retabler stak
    Pop    Bp
    Ret    10      2+4+4    fjern activationrecord
-----}
end;

```

```

Procedure WriteIntTabSum(Antal: Integer; Var Itab: ItabType);
Var
  Sum: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
  Push    Bp
  Mov     Bp, Sp
  Sub    Sp, 2           plads til lokal-var

Der er desuden oprettet følgende symboler:
Sum      Equ      [Bp-2]
Itab     Equ      [Bp+4]      adresse
Antal    Equ      [Bp+8]      Value
-----}

ASM
Mov    Ax, Ss: [Antal]   { IntTabSum(Antal, Itab, Sum); }
Push   Ax                { value Antal }
LEs   Bx, [Itab]         { adr. på Itab }
Push   Es
Push   Bx
Lea    Bx, [Sum]         { adr. på lokalvar. Sum }
Push   Ss
Push   Bx
Call  IntTabSum

Mov    Ax, Ss: [Sum]     { WriteInt(Sum); }
Push   Ax                { value Sum }
Call  WriteInt
END;
{-----
  pascal har her selv indskudt følgende epilog
  Mov    Sp, Bp           retabler stak
  Pop   Bp
  Ret   6     2+4        fjern activationrecord
-----}
end;

```

```

Procedure IntTabGsnit(Antal: Integer; Var Itab: ItabType;
                      Var Gsnit: Integer);
Var
  Wsum: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
    Push      Bp
    Mov       Bp, Sp
    Sub       Sp, 2           plads til lokal-var

  Der er desuden oprettet følgende symboler:
  Wsum     Equ      [Bp-2]
  Gsnit    Equ      [Bp+4]      adresse
  Itab     Equ      [Bp+8]      adresse
  Antal    Equ      [Bp+12]     Value
-----}
ASM
  Mov      Ax, Ss: [Antal]   { IntTabSum(Antal, Itab, Sum); }
  Push    Ax                 { value Antal }
  LEs    Bx, [Itab]         { adr. på Itab }
  Push    Es
  Push    Bx
  Lea     Bx, [WSum]        { adr. på lokalvar. WSum }
  Push    Ss
  Push    Bx
  Call   IntTabSum

  Mov      Ax, Ss: [WSum]   { Gsnit := Wsum DIV Antal; }
  Cwd
  Mov      Bx, Ss: [Antal]
  DIV
  LEs    Bx, [Gsnit]        { adr. på Gsnit }
  Mov      Es: [Bx], Ax

  END;
{-----
  pascal har her selv indskudt følgende epilog
    Mov      Sp, Bp          retabler stak
    Pop     Bp
    Ret     10                2+4+4    fjern activationrecord
-----}
end;

```

```

Procedure WriteIntTabGsnit(Antal: Integer; Var Itab: ItabType);
Var
  Gsnit: Integer;
begin
{-----
  pascal har her selv indskudt følgende prolog
    Push      Bp
    Mov       Bp, Sp
    Sub      Sp, 2           plads til lokal-var

Der er desuden oprettet følgende symboler:
Gsnit   Equ     [Bp-2]
Itab    Equ     [Bp+4]      adresse
Antal   Equ     [Bp+8]      Value
-----}

ASM
  Mov      Ax, Ss: [Antal]    { IntTabGsnit(Antal, Itab, Sum); }
  Push     Ax                { value Antal }
  LEs     Bx, [Itab]         { adr. på Itab }
  Push     Es
  Push     Bx
  Lea      Bx, [Gsnit]       { adr. på lokalvar. Gsnit }
  Push     Ss
  Push     Bx
  Call    IntTabGsnit

  Mov      Ax, Ss: [Gsnit]    { WriteInt(Gsnit); }
  Push     Ax                { value Gsnit }
  Call    WriteInt

END;
{-----
  pascal har her selv indskudt følgende epilog
    Mov      Sp, Bp          retabler stak
    Pop     Bp
    Ret     6                 2+4     fjern activationrecord
-----}
end;

```

Var

```
Tekst:      String;
Itab:       ItabType;
Antal:      Integer;
Sum:        Integer;
Gsnit:      Integer;
```

begin

```
Tekst := 'Beregning af sum og gennemsnit'#13#10; {ikke oversat}
ASM
```

```
Lea      Bx, [Tekst]      { WriteStr(Tekst); }
Push    Ds
Push    Bx
Call   WriteStr
END;
```

```
Tekst := 'Intast positivt heltal: ' ;           {ikke oversat}
ASM
```

```
Mov      Antal,2          { Antal := 2; }
END;
ASM
```

```
Lea      Bx, [Tekst]      { ReadIntTab(Tekst, Antal, Itab); }
Push    Ds
Push    Bx
Mov      Ax,Antal         { value antal }
Push    Ax
Lea      Bx, [Itab]        { adr. på Itab }
Push    Ds
Push    Bx
Call   ReadIntTab
END;
```

```
Tekst := 'Oversigt over tallene:'#13#10;           {ikke oversat}
ASM
```

```
Lea      Bx, [Tekst]      { WriteStr(Tekst); }
Push    Ds
Push    Bx
Call   WriteStr
END;
```

ASM

```
Mov      Ax,Antal         { WriteIntTab(Antal, Itab); }
Push    Ax
Lea      Bx, [Itab]        { value antal }
Push    Ds
Push    Bx
Call   WriteIntTab
END;
```

```
Tekst := 'Summen er: ' ;                   {ikke oversat}
ASM
```

```
Lea      Bx, [Tekst]      { WriteStr(Tekst); }
Push    Ds
Push    Bx
Call   WriteStr
END;
```

ASM

```
Mov      Ax,Antal         { WriteIntTabSum(Antal, Itab); }
Push    Ax
Lea      Bx, [Itab]        { value antal }
Push    Ds
Push    Bx
Call   WriteIntTabSum
END;
```

```
Tekst := 'Gennemsnittet er: ';           {ikke oversat}
ASM
    Lea      Bx, [Tekst]      { WriteStr(Tekst); }
    Push    Ds
    Push    Bx
    Call   WriteStr
END;

ASM
    Mov      Ax, Antal      { WriteIntTabGsnit(Antal, Itab); }
    Push    Ax
    Lea      Bx, [Itab]       { value antal
                                { adr. på Itab }
    Push    Ds
    Push    Bx
    Call   WriteIntTabGsnit
END;

ReadLN;                                {ikke oversat}

end.
```

7.5.c

Programeksempel i assembler

Bjørk Busch

Dette programeksempel bygger på det foregående pascalprogram, men implementeringen er istedet foretaget i assembler.

Programmet er optimeret med hensyn til tabeladressering ved sekventiel genemløb. Der anvendes desuden registre istedet for lokal-variable, hvor det er muligt.

```
; Program SumSnit

DATA      Segment Para Public 'DATA'
;=====
Tekst    Db      128 DUP (?)           ;1+127
Itab     Dw      100 DUP (?)
Antal    Dw      ?
Sum      Dw      ?
Gsnit    Dw      ?
Tekst1   Db      'Beregning af sum og gennemsnit',13,10,0
Tekst2   Db      'Intast positivt heltal: ',0
Tekst3   Db      'Oversigt over tallene:',13,10,0
Tekst4   Db      'Summen er: ',0
Tekst5   Db      'Gennemsnittet er: ',0
;=====
DATA      Ends               ; slut på data-segment

CODE      Segment Para Public 'CODE'
;=====
Assume Cs:CODE,Ds:DATA,Es:nothing,Ss:STACK

; Proceduren InitStr anvendes til initiering af streng fra
; strengkonstant
; Ds:Si -> nyt indhold termineret med binært nul
; Es:Di -> modtager streng
InitStr  Proc  Near
    Push   Ax
    Push   Di
    Push   Si
    Mov    Ah,0           ; Længde
StrLoop:
    Inc    Di             ; næste til tegn
    Mov    Al,Ds:[Si]
    Cmp    Al,0
    Je    StrLoopEnd
    Mov    Es:[Di],Al
    Inc    Ah             ; antal tegn + 1
    Inc    Si             ; Næste fra tegn
    Jmp    StrLoop
StrLoopEnd:
    Pop   Si
    Pop   Di
    Mov   Es:[Di],Ah
    Pop   Ax
    Ret
InitStr  EndP
```

```

;-----[  

Bin2Asc Proc Near ; (Ital: Integer; Var Stal: StalType);  

    aStal     Equ      [Bp+4]          ;adresse  

    aItal     Equ      [Bp+8]          ;value  

;-----[  

; prolog  

    Push     Bp  

    Mov      Bp, Sp  

    Sub     Sp, 0           ;plads til lokal-var  

;-----[  

    Push     Ax  

    Push     Bx  

    Push     Cx  

    Push     Dx  

    Push     Di  

    Push     Es  

    LEs      Di,Ss:[aStal]   ;Stal[0] := Chr(5);  

    Mov      Al,5  

    Mov      Es:[Di],Al  

    Mov      Cx,5           ;For I := 5 downto 1 do begin  

    Add      Di,5           ; Es:Di -> Stal[I]  

                           ;counter og index adskilles  

    Mov      Ax,Ss:[aItal]  

aForBegin:  

    Mov      Bx,10          ; Wtal := Ital MOD 10;  

    Cwd  

    Div      Bx             ; DxAx := Ax forbered 16bit division  

                           ; Ax:=Ital DIV 10, Dx:=Ital MOD 10  

    Add      Dl,'0'          ; Stal[I]:= Chr(Wtal+Ord('0'));  

    Mov      Es:[Di],Dl      ; Ital := Ital DIV 10; se DIV  

    Dec  

    Loop    aForBegin       ;end;  

                           ;counter auto, index manuel  

    Pop     Es  

    Pop     Di  

    Pop     Dx  

    Pop     Cx  

    Pop     Bx  

    Pop     Ax  

;-----[  

; epilog  

    Mov      Sp,Bp          ; retabler stak  

    Pop     Bp  

    Ret      6               ; 2+4      fjern activationrecord  

;-----[  

Bin2Asc EndP

```

```

WriteStr Proc Near ;(Var Streng: String);
;
    bStreng     Equ      [Bp+4]           ;adresse
;
; prolog
        Push      Bp
        Mov       Bp, Sp
        Sub      Sp, 0            ;plads til lokal-var
;
        Push      Ax
        Push      Cx
        Push      Dx
        Push      Si
        Push      Es

        LES      Si, Ss: [bStreng]   ; Antal := Length(Streng);
        Mov      Cl, Byte PTR Es: [Si]
        Mov      Ch, 0             ; Cx = antal

        Inc      Si                ; Es:Si => Streng[I]

        Cmp      Cx, 0            ; While Antal > 0 do begin
bWhileTst:
        Jle      bEndWhile
        Mov      Dl, Es: [Si]      ; Write(Streng[I]);
        Mov      Ah, 02
        Int      21h
        Inc      Si                ; I := I + 1;
        Dec      Cx                ; Antal := Antal - 1;
        Jmp      bWhileTst        ; end;

bEndWhile:
        Pop      Es
        Pop      Si
        Pop      Dx
        Pop      Cx
        Pop      Ax
;
; epilog
        Mov      Sp, Bp           ;retabler stak
        Pop      Bp
        Ret      4                 ;fjern activationrecord
;
WriteStr EndP

```

```

WriteInt Proc Near ; (Ital: Integer);
;-----
    cStal     Equ      [Bp-6]
    cItal     Equ      [Bp+4]           ;value
;-----
; prolog
    Push     Bp
    Mov      Bp, Sp
    Sub      Sp, 6           ;plads til lokal-var
;-----
    Push     Ax
    Push     Bx
    Push     Dx
    Push     Es

    Mov      Ax, [cItal]        ; Bin2Asc(Ital,Stal);
    Push     Ax
    Lea      Bx, [cStal]
    Push     Ss                 ; adr. på Stal
    Push     Bx
    Call    Bin2Asc

    Lea      Bx, [cStal]        ; WriteStr(Stal);
    Push     Ss                 ; adr. på Stal
    Push     Bx
    Call    WriteStr

    Mov      Ah, 2             ; WriteLN
    Mov      Dl, 13
    Int     21h
    Mov      Dl, 10
    Int     21h

    Pop     Es
    Pop     Dx
    Pop     Bx
    Pop     Ax
;-----
; epilog
    Mov      Sp, Bp           ; retabler stak
    Pop     Bp
    Ret     2                  ; fjern activationrecord
;-----
WriteInt EndP

```

```

WriteIntTab  Proc  Near      ;(Antal: Integer; Var Itab: ItabType);
;
    dItab      Equ      [Bp+4]           ;adresse
    dAntal     Equ      [Bp+8]           ;value
;
; prolog
    Push      Bp
    Mov       Bp, Sp
;
    Push      Ax
    Push      Cx
    Push      Si
    Push      Es

    LES       Si,Ss:[dItab]          ; I := 1; Es:Si=>Itab[1]

    Mov       Cx,Ss:[dAntal]        ; While Antal > 0 do begin
    Cmp       Cx, 0
dWhileTst:
    Jle       dEndWhile
    Mov       Ax,Es:[Si]           ; Ital := Itab[I];
    Push      Ax
    Call      WriteInt
    Add       Si, 2               ; I := I + 1;
    Dec       Cx                 ; Antal := Antal - 1;
    Jmp       dWhileTst          ; end;
dEndWhile:
    Pop      Es
    Pop      Si
    Pop      Cx
    Pop      Ax
;
; epilog
    Mov      Sp,Bp              ;retabler stak
    Pop      Bp
    Ret      6                  ;fjern activationrecord
;
WriteIntTab  EndP

```

```

Asc2Bin  Proc  Near ; (Var Stal: StalType; Var Ital: Integer);
;-----[-----]
    eItal      Equ      [Bp+4]          ;adresse
    eStal      Equ      [Bp+8]          ;adresse
;-----[-----]
; prolog
    Push      Bp
    Mov       Bp,Sp
    Sub       Sp,8           ;plads til lokal-var
;-----[-----]
    Push      Ax
    Push      Bx
    Push      Cx
    Push      Dx
    Push      Di
    Push      Si
    Push      Es

    LES      Si,Ss:[eStal]   ; Es:Si=>Stal[0]
    Mov      Cl,Byte PTR Es:[Si] ; L := Ord(Stal[0]);
    Mov      Ch,0

    Inc      Si           ; Es:Si=>Stal[1]

    Mov      Ax,0         ; Wtall1 := 0;

    Mov      Bx,10
    Mov      Dh,0;

eForTst:
    Cmp      Cx,0         ; For I := 1 to L do begin
    Jle      eEndFor
    Mul      Bx           ; Wtall1 := Wtall1 * 10;

    Mov      Dl,Es:[Si]   ; Wtall2 := Ord(Stal[I])
    Sub      Dl,'0'        ; - Ord('0');
    Add      Ax,Dx        ; Wtall1 := Wtall1+Wtall2;

    Inc      Si
    Dec      Cx           ; end;

    Jmp      eForTst

eEndFor:
    LES      Bx,Ss:[eItal] ; Ital := Wtall1;
    Mov      Es:[Bx],Ax

    Pop      Es
    Pop      Si
    Pop      Di
    Pop      Dx
    Pop      Cx
    Pop      Bx
    Pop      Ax
;-----[-----]
; epilog
    Mov      Sp,Bp      ; retabler stak
    Pop      Bp
    Ret      8           ; 4+4     fjern activationrecord
;-----[-----]
Asc2Bin  EndP

```

```

ReadInt  Proc  Near   ;(Var Tekst: String; Var Ital: Integer);
;-----[-----]
    fBuf      Equ      [Bp-10]
    fBufxBufLen Equ      [Bp-10]
    fBufxBufArr Equ      [Bp-09]
    fWtal     Equ      [Bp-2]
    fItal     Equ      [Bp+4]           ;adresse
    fTekst    Equ      [Bp+8]           ;adresse
;-----[-----]
; prolog
    Push     Bp
    Mov      Bp, Sp
    Sub      Sp, 10          ;plads til lokal-var
;-----[-----]
    Push     Ax
    Push     Bx
    Push     Dx
    Push     Es

    LEs     Bx, [fTekst]      ; WriteStr(Tekst);
    Push     Es               ; adr. på Tekst
    Push     Bx
    Call    WriteStr

    Push     Ds               ; ReadLN(Stal)
    Mov      Ax, Ss
    Mov      Ds, Ax
    Mov      Al, 6
    Mov      Ss: [fBufxBuflen], Al
    Lea      Dx, [fBuf]
    Mov      Ah, 0Ah
    Int     21h
    Mov      Dl, 10
    Mov      Ah, 2
    Int     21h
    Pop     Ds

    Lea      Bx, [fBufxBufarr] ; Asc2Bin(Buffarr, Wtal);
    Push     Ss               ; adr. på lokalvar. BufArr
    Push     Bx
    Lea      Bx, [fWtal]       ; adr. på lokalvar Wtal
    Push     Ss
    Push     Bx
    Call    Asc2Bin

    Mov      Ax, Ss: [fWtal]   ; Ital := Wtal
    LEs     Bx, [fItal]
    Mov      Es: [Bx], Ax

    Pop     Es
    Pop     Dx
    Pop     Bx
    Pop     Ax
;-----[-----]
; epilog
    Mov      Sp, Bp          ; retabler stak
    Pop     Bp
    Ret     8                ; 4+4    fjern activationrecord
;-----[-----]
ReadInt  EndP

```

```

ReadIntTab Proc Near ;(Var Tekst: String; Antal: Integer;
                  ; Var Itab: ItabType);
;-----
    gItal      Equ     [Bp-4]
    gI         Equ     [Bp-2]
    gItab     Equ     [Bp+4]           ;adresse
    gAntal    Equ     [Bp+8]           ;value
    gTekst    Equ     [Bp+10]          ;adresse
;-----
; prolog
    Push     Bp
    Mov      Bp, Sp
    Sub      Sp, 4           ;plads til lokal-var
;-----
    Push     Ax
    Push     Bx
    Push     Cx
    Push     Di
    Push     Es

    LEs     Di, Ss: [gItab]       ; I := 1;
            ; Es:Di -> Itab[I]
    Mov      Cx, Ss: [gAntal]
    Cmp      Cx, 0             ; While Antal > 0 do begin
gWhileTst:
    Jle     gEndWhile

    Push     Cx           ; save før kald
    Push     Di           ; kan undværes da
    Push     Es           ; underprog. reetabler

    LEs     Bx, [gTekst]       ; ReadInt(Tekst, Ital);
    Push     Es           ; adr. på Tekst
    Push     Bx
    Lea     Bx, [gItal]
    Push     Ss           ; adr. på lokalvar. Ital
    Push     Bx
    Call    ReadInt

    Pop     Es           ; restore efter kald
    Pop     Di           ; kan undværes da
    Pop     Cx           ; underprog. reetabler

    Mov     Ax, Ss: [gItal]       ; Itab[I] := Ital;
    Mov     Es: [Di], Ax

    Add     Di, 2           ; I := I + 1;
    Dec     Cx           ; Antal := Antal - 1;

    Jmp     gWhileTst        ; end;
gEndWhile:
    Pop     Es
    Pop     Di
    Pop     Cx
    Pop     Bx
    Pop     Ax
;-----
; epilog
    Mov     Sp, Bp          ; retabler stak
    Pop     Bp
    Ret     10              ; 4+2+4   fjern activationrecord
;-----
ReadIntTab EndP

```

```

IntTabSum  Proc  Near ; (Antal: Integer; Var Itab: ItabType;
;                           Var Sum: Integer);
;-----
    hWsum      Equ      [Bp-4]
    hI         Equ      [Bp-2]
    hSum       Equ      [Bp+4]           ;adresse
    hItab      Equ      [Bp+8]           ;adresse
    hAntal     Equ      [Bp+12]          ;Value
;-----
; prolog
    Push      Bp
    Mov       Bp, Sp
    Sub       Sp, 4           ; plads til lokal-var
;-----
    Push      Ax
    Push      Bx
    Push      Cx
    Push      Di
    Push      Si
    Push      Es

    LES       Si,Ss:[hItab]      ; I := 1;
              ; Es:Si -> Itab[I]
    Mov       Ax, 0            ; Wsum := 0;
    Mov       Cx,Ss:[hAntal]
    Cmp       Cx, 0            ; While Antal > 0 do begin
hWhileTst:
    Jle      hEndWhile

    Add       Ax,Es:[Si]        ; Wsum:= Wsum + Itab[I];
    Add       Si, 2             ; I := I + 1;
    Dec       Cx                ; Antal := Antal - 1;
    Jmp      hWhileTst        ; end;
hEndWhile:
    LES       Bx,Ss:[hSum]      ; Sum := Wsum;
    Mov       Es:[Bx],Ax

    Pop      Es
    Pop      Si
    Pop      Di
    Pop      Cx
    Pop      Bx
    Pop      Ax
;-----
; epilog
    Mov      Sp,Bp           ; retabler stak
    Pop      Bp
    Ret      10               ;2+4+4  fjern activationrecord
;-----
IntTabSum  EndP

```

```

WriteIntTabSum Proc NEAR ;(Antal: Integer; Var Itab: ItabType);
;-----
    iSum      Equ      [Bp-2]
    iItab     Equ      [Bp+4]          ;adresse
    iAntal    Equ      [Bp+8]          ;Value
;-----
; prolog
    Push     Bp
    Mov      Bp, Sp
    Sub      Sp, 2           ;plads til lokal-var
;-----
    Push     Ax
    Push     Bx
    Push     Es

    Mov      Ax, Ss:[iAntal]       ; IntTabSum(Antal, Itab, Sum) ;
    Push     Ax                  ; value Antal
    LES     Bx, [iItab]          ; adr. på Itab
    Push     Es
    Push     Bx
    Lea      Bx, [iSum]          ; adr. på lokalvar. Sum
    Push     Ss
    Push     Bx
    Call    IntTabSum

    Mov      Ax, Ss:[iSum]       ; WriteInt(Sum) ;
    Push     Ax                  ; value Sum
    Call    WriteInt

    Pop     Es
    Pop     Bx
    Pop     Ax
;-----
; epilog
    Mov      Sp, Bp      ;      retabler stak
    Pop     Bp
    Ret      6          ; 2+4      fjern activationrecord
;-----
WriteIntTabSum EndP

```

```

IntTabGsnit Proc Near ;(Antal: Integer; Var Itab: ItabType;
;                           Var Gsnit: Integer);
;-----
    jWsum      Equ      [Bp-2]
    jGsnit     Equ      [Bp+4]           ; adresse
    jItab      Equ      [Bp+8]           ; adresse
    jAntal     Equ      [Bp+12]          ; Value
;-----
; prolog
    Push      Bp
    Mov       Bp, Sp
    Sub       Sp, 2            ; plads til lokal-var
;-----
    Push      Ax
    Push      Bx
    Push      Dx
    Push      Es

    Mov       Ax, Ss : [jAntal]      ; IntTabSum(Antal, Itab, Sum) ;
    Push      Ax                   ; value Antal
    LEs      Bx, [jItab]          ; adr. på Itab
    Push      Es
    Push      Bx
    Lea       Bx, [jWSum]          ; adr. på lokalvar. WSum
    Push      Ss
    Push      Bx
    Call     IntTabSum

    Mov       Ax, Ss : [jWSum]      ; Gsnit := Wsum DIV Antal;
    Cwd
    Mov       Bx, Ss : [jAntal]
    DIV
    LEs      Bx, [jGsnit]          ; adr. på Gsnit
    Mov       Es : [Bx], Ax

    Pop      Es
    Pop      Dx
    Pop      Bx
    Pop      Ax
;-----
; epilog
    Mov       Sp, Bp      ;      retabler stak
    Pop      Bp
    Ret      10        ;2+4+4  fjern activationrecord
;-----
IntTabGsnit EndP

```

```

WriteIntTabGsnit Proc Near ;(Antal: Integer; Var Itab: ItabType);
;
    kGsnit      Equ      [Bp-2]
    kItab       Equ      [Bp+4]           ;adresse
    kAntal      Equ      [Bp+8]           ;Value
;
; prolog
    Push      Bp
    Mov       Bp, Sp
    Sub       Sp, 2            ;plads til lokal-var
;
    Push      Ax
    Push      Bx
    Push      Es
;
    Mov       Ax, Ss: [kAntal]   ; IntTabGsnit(Antal, Itab, Sum);
    Push      Ax             ; value Antal
    LEs      Bx, [kItab]       ; adr. på Itab
    Push      Es
    Push      Bx
    Lea       Bx, [kGsnit]     ; adr. på lokalvar. Gsnit
    Push      Ss
    Push      Bx
    Call     IntTabGsnit
;
    Mov       Ax, Ss: [kGsnit]   ; WriteInt(Gsnit);
    Push      Ax             ; value Gsnit
    Call     WriteInt
;
    Pop      Es
    Pop      Bx
    Pop      Ax
;
; epilog
    Mov      Sp, Bp          ; retabler stak
    Pop      Bp
    Ret      6               ;2+4    fjern activationrecord
;
WriteIntTabGsnit EndP

```

```

; Proceduren InitDs opsætter adresse på databasegmentet
InitDS  Proc   Near
        Mov     Ax, Seg DATA          ; etabler
        Mov     Ds, Ax              ; databasegment
        Ret                ; retur fra procedure
InitDS  EndP

; Proceduren StopPG overgiver kontrollen til DOS igen
StopPG  Proc   Near
        Mov     Al, 0              ; returkode = 0
        Mov     Ah, 4Ch            ; opsæt funktion for
        Int     21h               ; terminate og udfør
; Da denne funktion ikke giver kontrollen tilbage
; er der ingen Ret-instruktion
StopPG  EndP

```

```

; Proceduren Main er den rutine der får kontrollen ved start
Main    Proc   Near
        Call    InitDS           ; udfør InitDS

        Mov     Ax,Ds
        Mov     Es,Ax
        Lea     Di,Tekst
        Lea     Si,Tekst1
        Call   InitStr

        Lea     Bx,[Tekst]       ; WriteStr(Tekst);
        Push   Ds               ; adr. på tekst
        Push   Bx
        Call   WriteStr

        Mov     Ax,Ds
        Mov     Es,Ax
        Lea     Di,Tekst
        Lea     Si,Tekst2
        Call   InitStr

        Mov     Antal,2

        Lea     Bx,[Tekst]       ; ReadIntTab(Tekst, Antal, Itab);
        Push   Ds               ; adr. på tekst
        Push   Bx
        Mov     Ax,Antal         ; value antal
        Push   Ax
        Lea     Bx,[Itab]         ; adr. på Itab
        Push   Ds
        Push   Bx
        Call   ReadIntTab

        Mov     Ax,Ds
        Mov     Es,Ax
        Lea     Di,Tekst
        Lea     Si,Tekst3
        Call   InitStr

        Lea     Bx,[Tekst]       ; WriteStr(Tekst);
        Push   Ds               ; adr. på tekst
        Push   Bx
        Call   WriteStr

        Mov     Ax,Antal         ; WriteIntTab(Antal, Itab);
        Push   Ax               ; value antal
        Lea     Bx,[Itab]         ; adr. på Itab
        Push   Ds
        Push   Bx
        Call   WriteIntTab

        Mov     Ax,Ds
        Mov     Es,Ax
        Lea     Di,Tekst
        Lea     Si,Tekst4
        Call   InitStr

        Lea     Bx,[Tekst]       ; WriteStr(Tekst);
        Push   Ds               ; adr. på tekst
        Push   Bx
        Call   WriteStr

```

```

Mov      Ax,Antal           ; WriteIntTabSum(Antal,Itab) ;
Push     Ax                 ; value antal
Lea      Bx,[Itab]          ; adr. på Itab
Push     Ds
Push     Bx
Call    WriteIntTabSum

Mov      Ax,Ds
Mov      Es,Ax
Lea      Di,Tekst
Lea      Si,Tekst5
Call   InitStr

Lea      Bx,[Tekst]         ; WriteStr(Tekst) ;
Push     Ds                 ; adr. på tekst
Push     Bx
Call   WriteStr

Mov      Ax,Antal           ; WriteIntTabGsnit(Antal,Itab) ;
Push     Ax                 ; value antal
Lea      Di,[Itab]          ; adr. på Itab
Push     Ds
Push     Di
Call   WriteIntTabGsnit

Mov      Ah,01
Int     21h                 ; vent på tast

Main   Call   StopPG        ; udfør StopPG
      EndP

;=====
CODE   EndS                ; slut på code-segment

STACK  Segment Para Stack 'STACK'
;=====
dw      128 dup(?)         ; der er sat 128 ord af
;=====
STACK  EndS                ; slut på stack-segment

End    Main                 ; Slut og opsæt startadr.

```